Lecture - 112 :

String, StringBuffer & StringBuilder for Java Certification.

(1) Difference b/w String & StringBuffer.

Case (1) String : String is immutable. means String Object is immutable (Not-changeable).

Once we create an String object we can't perform any changes in String object. If any person trying to perform changes with those changes a new object will be created. This non-changeable behaviour is nothing but immutability.

→ In the existing object we can't perform any changes.

StringBuffer : StringBuffer object is mutable. Once we create StringBuffer object we can perform any changes in StringBuffer object, so this changeable behaviour is nothing but mutability.

eg: immutability.

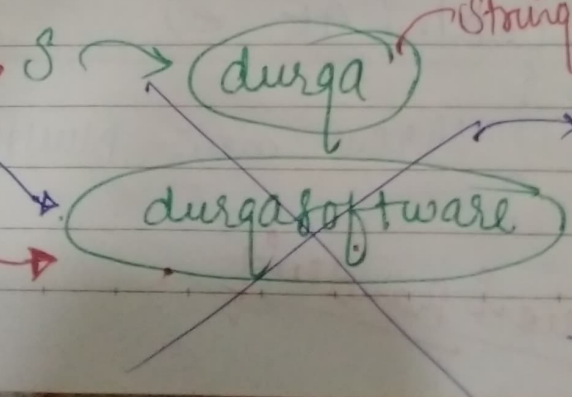
```
String s = new String("durga");
s.concat("software");
sopln(s);
```

O/p : durga

Still s is pointing to durga.

I'm not assigning any reference variable to this new object

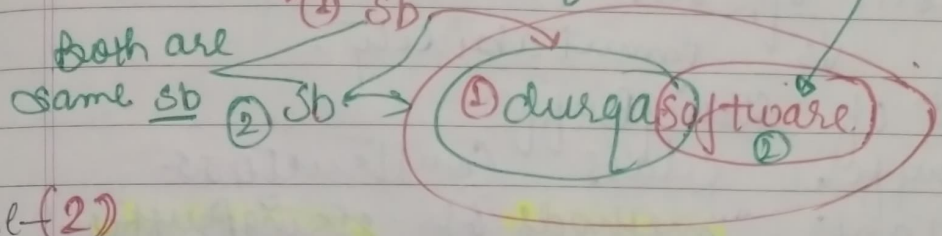
With those changes a new object is created i.e.



mahak ™ Date: Page:

eg:- mutability

① StringBuffer sb = new StringBuffer("durga");
② sb.append("software");
sopln(sb);
O/P :- durgasoftware.



Case-(2)

(2) == Operator Vs equals() method :-

Next Page

→ Topics:-

① Difference b/w String & StringBuffer.

Case 1 * mutability Vs Immutability
Case 2 * equals() method.

② String object creation : Heap & SCP.
③ Importance of String constant Pool.
4) Important FAQs on String & StringBuffer.
5) Important Constructors of String class.
6) Important methods of String class:

(10)
* charAt()
* equals()
* isEmpty()
* replace()
* indexOf()
* concat()
* equalsIgnoreCase()
* length()
* substring()
* lastIndexOf()

* toLowerCase()
* trim()
* toUpperCase()

7) Important conclusion about string immutability.

8) Creation of our Immutable class
9) final Vs immutability.
10) Need of StringBuffer.
11) StringBuffer class constructors.
12) Important methods of StringBuffer.

(12)
* length()
* charAt()
* append()
* delete()
* reverse()
* ensureCapacity()
* capacity()
* setCharAt()
* insert()
* deleteCharAt()
* setLength()
* trimToSize()

13) Need of StringBuilder.
14) Differences b/w StringBuffer and StringBuilder
15) String Vs StringBuffer Vs StringBuilder
16) Method chaining.

---

(2)

Case 2:

Imp ** == operator Vs equals method() :

T

== Operator: In the case of String & .equals() methods:

Def:

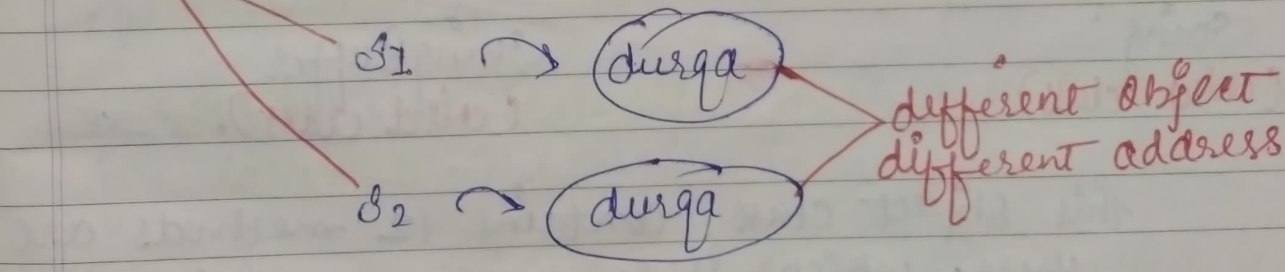
== operator always meant for reference comparision. Reference comparision means if both references pointing to the same object then only it is going to Returns true.

deals.

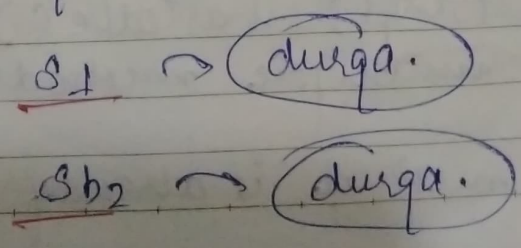
of object address.

If Reference pointing to different objects then automatically returns false.

```
String s1 = new String("durga");
String s2 = new String("durga");
sopln(s1==s2); false.
sopln(s1.equals(s2));
```



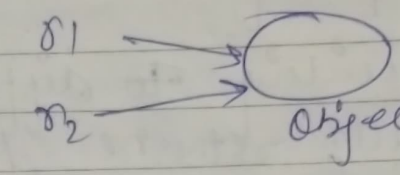

```
String sb1 = new StringBuffer("durga");
String sb2 = new StringBuffer("durga");
sopln(sb1==sb2); false.
sopln(sb1.equals(sb2));
```



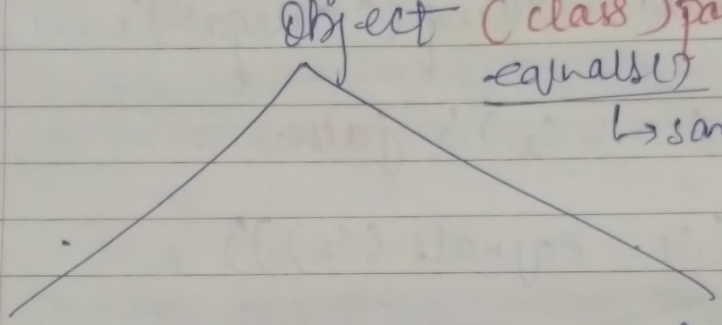
if $r_1 == r_2$ ⇒ true.

if r_1 & r_2 pointing to same object.

eg:



• equals() method :-



This Object class contains 12 methods are there which is commonly required for every thing. ofcourse one method is private. so we don't require much importance. By default 11 methods are there.

Object class contains 11 methods which are required for every class and object. In the toString(), hashCode(), equals() method & multiple methods are there.

equals() method is already available for.

Object class :-

Object class equals() method meant for reference/address comparision.

* If both reference pointing to same object then only equals() method returns true. Otherwise false. So it is exactly same as == operator.

* So, by default equals() method present in Object class are meant for reference comparision not for content comparision

⇒ But, Now in the child classes equals method is overriden for content comparision.

String :-

In String class equals() method is overriden for content comparision.

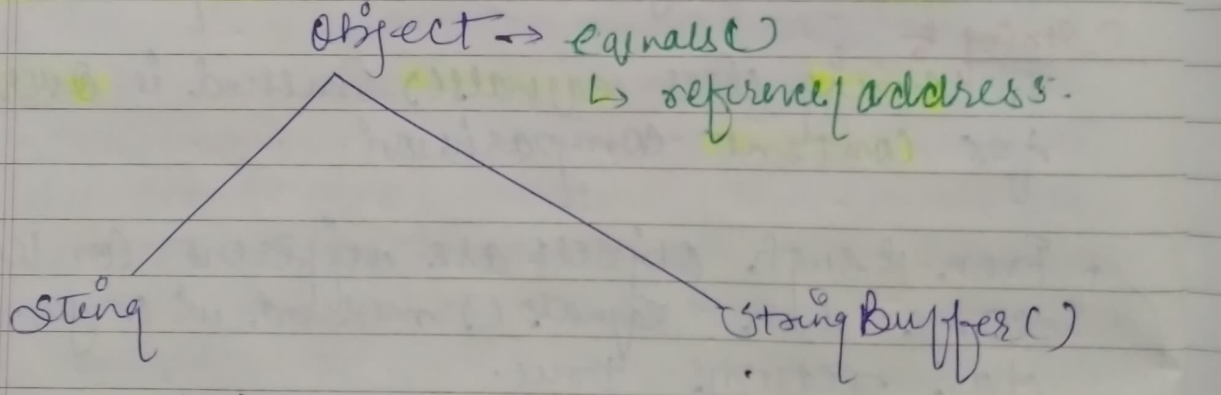
Even though objects are different Content is same then equals() method in or for String class returns true.

So in String class if i can apply a equals() method for String object. it is always meant for content comparision only. because in String class equals() method is overriden for content comparision.

Note:- StringBuffer :-

In StringBuffer the equals() method is not overriden. if it is not overriden if any person calling on the StringBuffer object equal method, then which equals() will be called? Object class equals() will be called. Object class equals() always meant for reference comparision.

★ The equals() method present in object class also meant for reference comparision. we have to override in a class child. for content comparision for our requirement.



eg :-

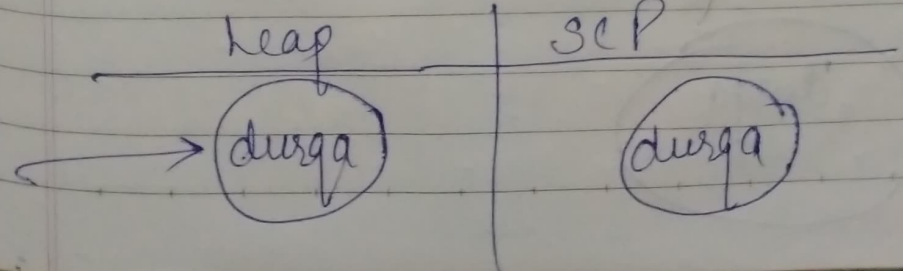
```
String s1 = new String("durga");
String s2 = new String("durga");
sopln(s1==s2); false
sopln(s1.equals(s2)); true
```

```
StringBuffer sb1 = new StringBuffer("durga");
StringBuffer sb2 = new StringBuffer("durga");
sopln(sb1==sb2); false
sopln(sb1.equals(sb2)); false
```

Conclusion:
In String equals method always meant for content comparision.
But in StringBuffer equals method meant for reference comparision.

(2) String Object creation: Heap and String Constant Pool (SCP) Part-1.

```
String s = new String("durga");
```



By using new keyword. Once object is created in Heap area.
And, for every String literal once object will be created in SCP area. (String Constant Pool).

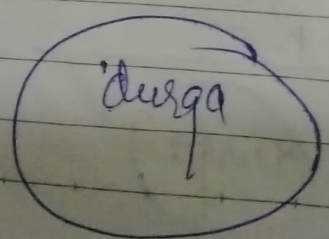
In SCP area object no explicit reference variable by implicit reference variable will be maintain by JVM. That's why we need only for this reusing same object for future purpose. and this object is not eligible for G.C. bcz implicitly reference variable is maintained by JVM.

If any object is created by String literal this object is not eligible for G.C. bcz JVM.

Until 1.6 V SCP is a part of Method area PERMGEN.
But, 1.7V onwards for efficient memory utilization. this SCP area moved to Heap area.

② String s = "durga";

SCP (Area)

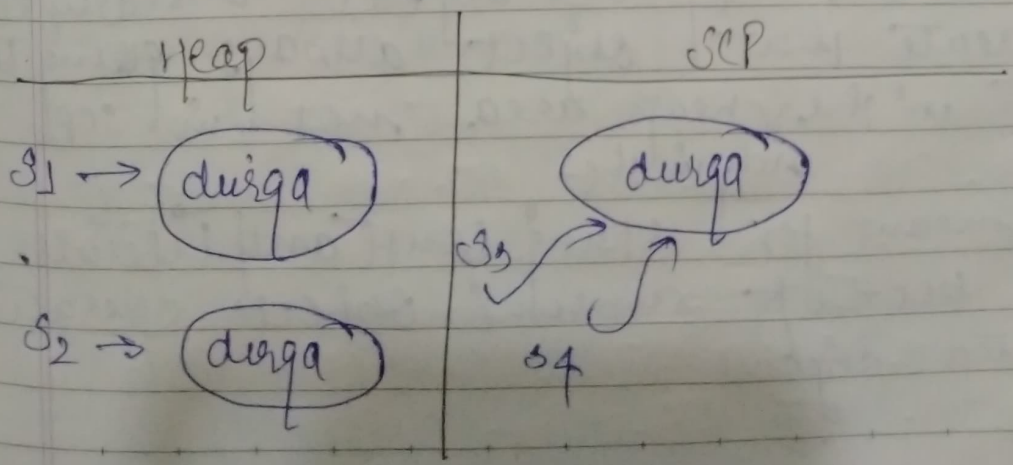


In this case means first JVM check is there any object with the same content already available in SCP or not. if it is already there then it will use the same object. if the object is not present then only the new object is going to create and S is pointing to that object.

Anyway object creation in the heap area is always mandatory whenever we are using new object compulsory new object will be created in heap area.
But if we are going to create like String constant literal then in this case first JVM will check for existance of that object with the same content. if not then create new ~~copy of heap~~ object.

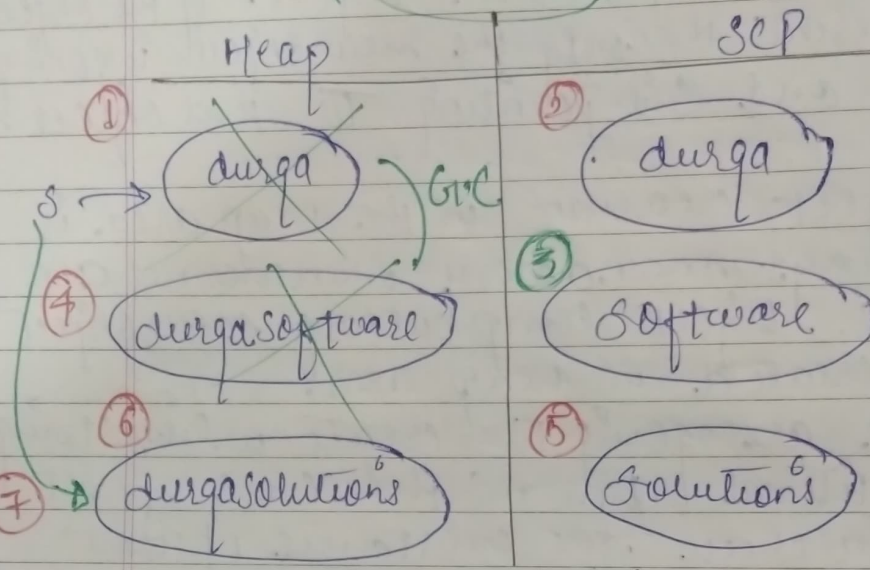
(3)

eg:- String s1 = new String("durga");
String s2 = new String("durga");
String s3 = "durga";
String s4 = "durga";



Eg:2:- String s = new String ("durga");
s.concat ("software");
s.concat ("solutions");

no reference var → s = s.concat ("solutions");
ref var → result



Object created ⇒ 6 GC ⇒ 2 SCP ⇒ 3
sopln(s); → durga.solutions Heap → 3

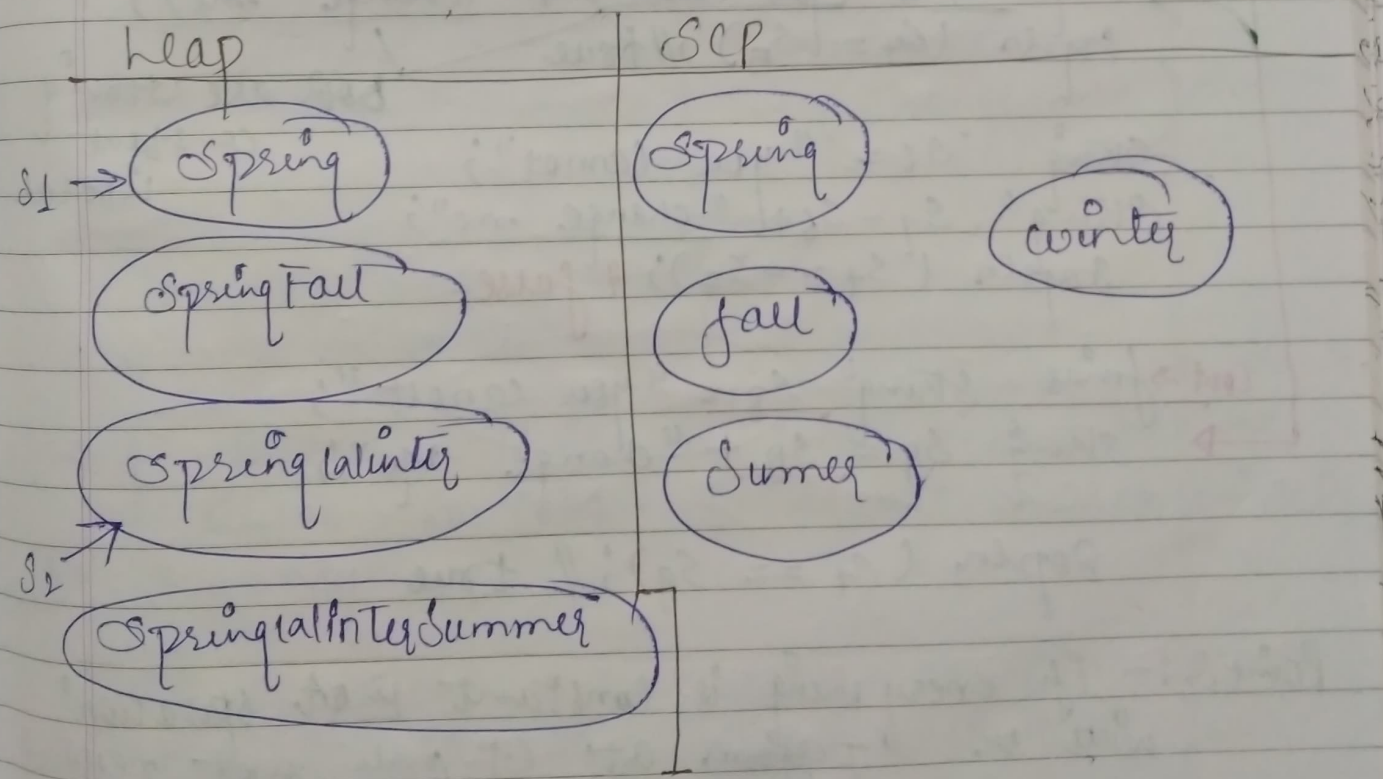
→ String is immutable if we are trying to perform change of content then we can't change the content. If we are trying to perform the change compulsory a new object is going to created because of runtime operation may be method call. If an object is required to create that object always going to create in the heap area not in scp only in the

SCP meant for String constant literals. not because of runtime object operation created objects.

Because of Runtime operation if the object will be created then that object will always be created on Heap area.

* Heap and String Constant Pool (SCP) Part-2

```
String s1 = new String("Spring");
s1.concat("Fall");
String s2 = s1.concat("winter");
s2.concat("Summer");
System.out.println(s1); Spring
System.out.println(s2); SpringWinter
```



Heap & String Constant Pool (SCP) Part-3 :-

① String s1 = new String ("you cannot change me");
② String s2 = new String ("you cannot change me");

Sopln (s1 == s2); // false

String s3 = "you cannot change me";
Sopln (s1 == s3); // false.

String s4 = "you cannot change me";
Sopln (s3 == s4); // true.

String s5 = "you cannot " + "change me";
Sopln (s4 == s5); // true

Both are String constant literal.

String s6 = "you cannot";
String s7 = s6 + "change me";
Sopln (s4 == s7); // false.

Cont → final String s8 = "you cannot ";
String s9 = s8 + "change me";

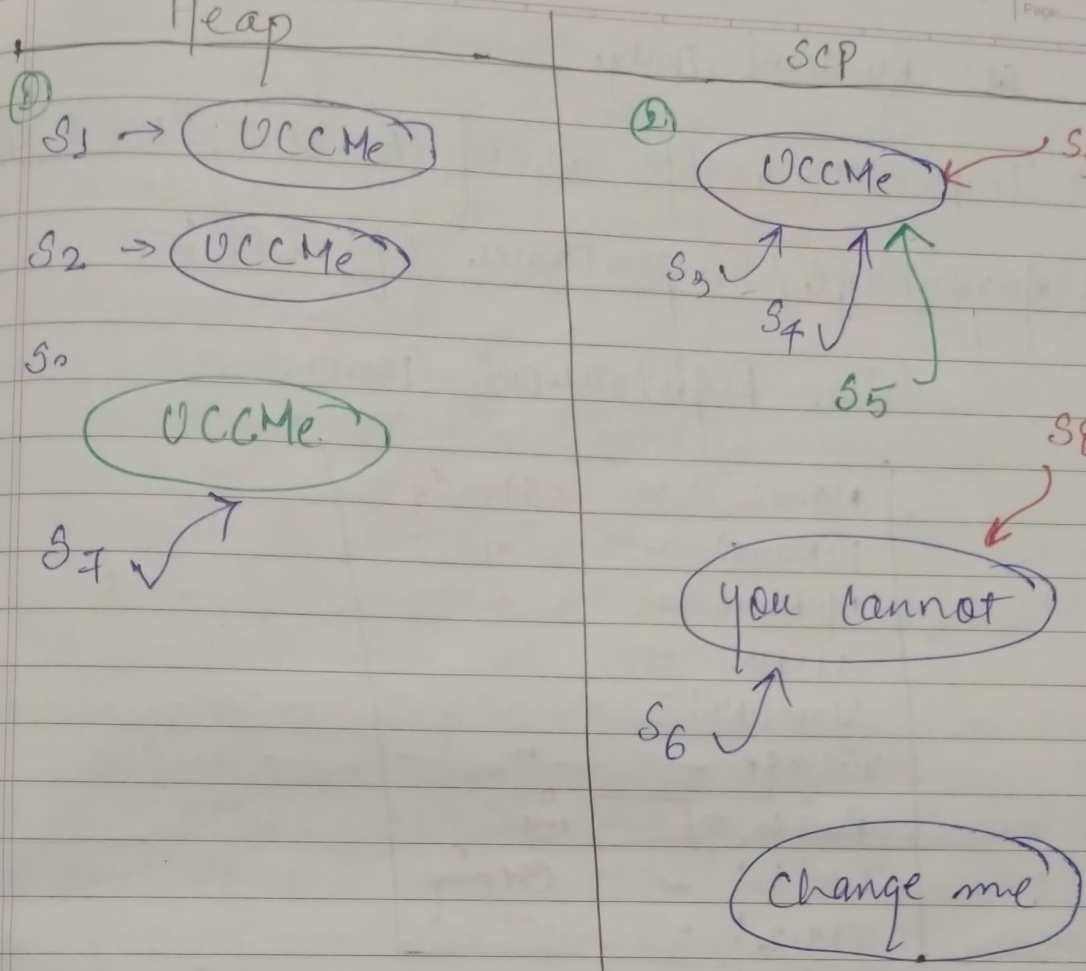
Sopln (s4 == s9); // true

Same as

Point s5 :- If everything is constant then operation will be perform at CT only not required to wait until Runtime.

So Both are Constant ∴ Concatenation going to happen at CT only.

Point s6



→ At Runtime this S_5 content will become literal. So JVM will always search in SCP area only.

This operation only perform at C.T because both arguments are constant.

Point S_6 & S_7 :— S_6 here is normal Variable + Content. If atleast one argument is a normal Variable, then this operation will be performed

at Runtime Only.

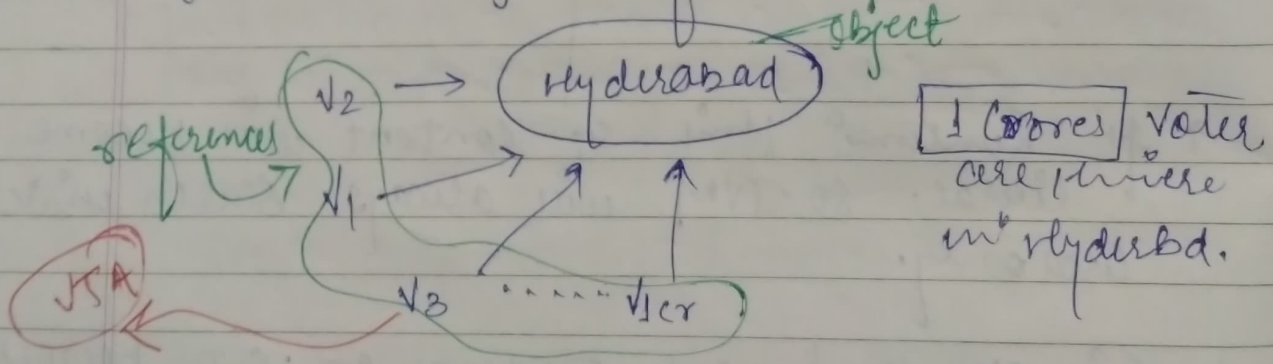
(3) Importance Of String Constant Pool (SCP) :-

* Advantage / Importance Of SCP :-

Voter Registration Form

Name:	String
F.Name:	"
M.Name	"
H.NO:	"
Street NO:	"
Village	"
Pin:	int
IM 1:	String
IM 2:	"

I want to fill my voter detail.



Create one object and share this one object to all 1 crore members. In Java it is possible only due to SCP objects.

In SCP only one object is created and

reused by everyone.

Advantage of SCP:-

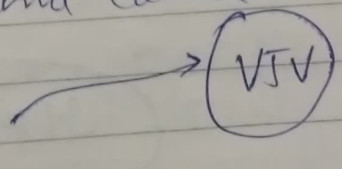
(1) Same object in SCP. will be referenced with multiple references. memory utilization will be improved by default. and performance will be improved.

Disadvantage of reference nature of SCP:-

If vatu V_3 want to change his city name and if V_3 open online app and change his name then all references going to affect. because multiple references reference one object.

So, after realizing this problem Java people analyse this and come with immutability Concept means. Once we create an string Object we are not allow to change it.

So, if any person wants to change its content then by this change a new object will be created and called immutability.



Why immutability is required? Just only there To SCP Concept. if this is not there then no requirement.

Merits of SCP:-
Same object can be reused. So, memory will be saved & performance will be improved.

Demerit of SCP:-
If by one reference we have change object content then all reference attached with definitely effect. So to prevent from this immutability concept introduced.

(4) * Important FAQs on String & StringBuffer:-

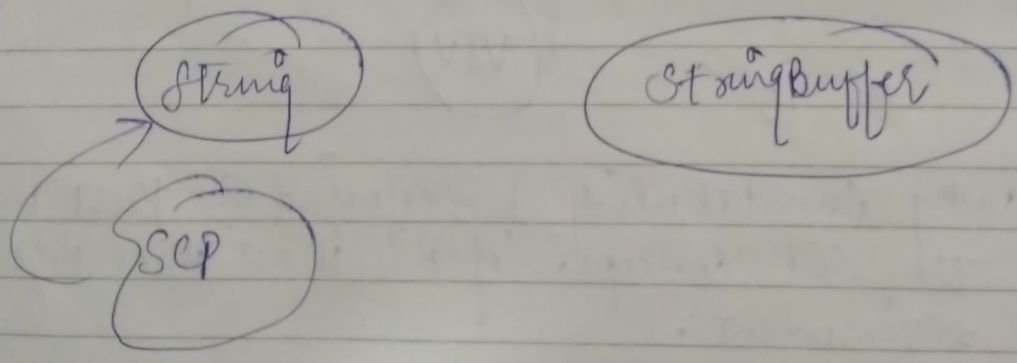
(1) Why SCP concept is available only for String object but not for StringBuffer?

(2) Why String objects are immutable where as StringBuffer objects are mutable?

(3) Inaddition to String objects any other objects are immutable in Java?

Ans 2:-

Ans 1:-



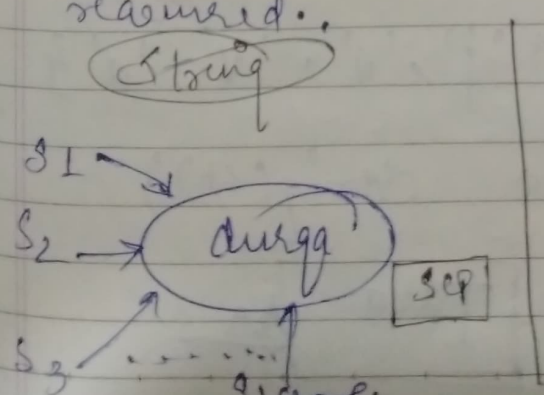
The most commonly used object in Java

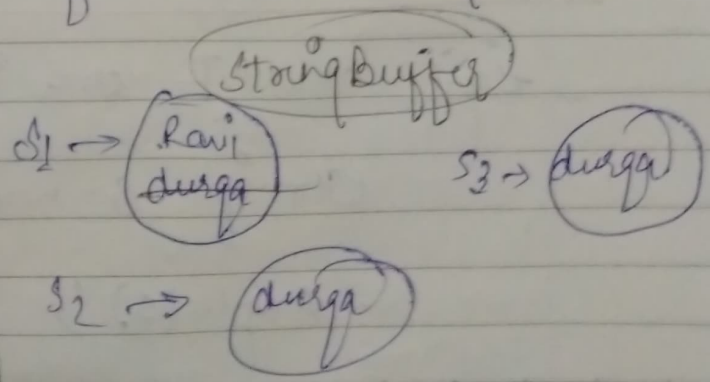
it String, so, special memory management provided by Java people for String.
But, String Buffer for 20K line of code you may use.

String is regular Customer to the bar, but not StringBuffer.

Ans27 For the string object SCP concept is available reusing the same object is there multiple references pointing to the same object. by using one reference if we are allow to change the content then the remaining references will be affected then to prevent that immutability is required.

But, here every reference separate object is there, because reusing the same object concept is not there bcoz SCP is not there. that's why by using one reference if we are allow to change the content then there is no affect on remaining object content. that's why immutability is not required.





Ans: All wrapper class objects are also immutable. by default.

eg: Byte, Short, Integer, Long, Float, Character, Double, Boolean.

(5) Important Constructors of String class:

① String s=new String();
→ creates an empty string object in an heap area.

② String s=new String(String literal);

③ String s=new String(StringBuffer sb);

④ String s=new String(StringBuilder sb);

⑤ String s=new String(char[] ch);

⑥ String s=new String(byte[] b);

for the given byte array we can create an equivalent String object.

eg: char[] ch={'j','a','v','a'};
String s=new String(ch);
sopln(s);

Java

· byte[] b = {97, 98, 99, 100} -128 to 127

String s = new String(b);

Sopln(s); abcd

(6) Important methods of String class:

① public char charAt(int index)

eg: String s = "durga";

Sopln(s.charAt(3)); q ✓

Sopln(s.charAt(30)); RE: StringIndexOutOfBoundsException

② public String concat(String s)

String s = "durga";

s = s.concat("software");

s → durga

Now → durgaSoftware

Sopln(s); → durgaSoftware.

runtime method.

alternative

s = s + "software";

s+ = "software";

Overloaded + & += operator also for concatenation only.

(3) public boolean equals(String s) (5)
or
(Object o)

→ To check equality of any String Objects.

String s = "DURGA";
Sopln(s.equals("durga"));
Op: false → bcz, Consider case also.

(4) public boolean equalsIgnoreCase(String s)
specially designed method. To ignore case

→ where case is ignored.

Eg: String s = "DURGA";
Sopln(s.equals("durga")); false
Sopln(s.equalsIgnoreCase("durga")); → true

App(1)
[Username] → equalsIgnoreCase()
[pwd] → equals()

(5) isEmpty()
(6) length()
(7) replace()
(8) substring()

(5) public boolean isEmpty().

eg1: String s = " ";
↳ no space
sopln(s.isEmpty()); true

eg2: String s = "durga";
sopln(s.isEmpty()); false.

(6) public int length().

eg: String s = "durga";
sopln(s.length()); 5

→ int[] x = {10, 20, 30, 40};
sopln(x.length); 4

Emp String [sopln(s.length); → CE. (length: var)]

Array [sopln(x.length()); → CE. (length(): method)]

(7) replace()

public String replace(char old, char new);

String s = "ababab";

sopln(s.replace('a','b')); bbbbbbb

(8) substring() :-

public string substring(int begin); (small)
index

substring ✗
Substring ✓

→ return string from begin index to end of string.

String s = "abcdefg";
0 1 2 3 4 5 6
sopln(s.substring(3)); defg

(9) public string substring(int begin, int end); (12)

from begin index to [end-1] index

sopln(s.substring(3,6)); def

(10) indexOf()
(11) lastIndexOf()
12) toLowerCase()
13) toUpperCase()

(10) public int indexOf(char ch);

returns index of specified character

eg: String s = "durga";
sopln(s.indexOf('g')); 3

→ sopln(s.indexOf('z')); −1 character NA

Eg: String s = "babbab";

sopln(s.indexOf('a')); 1

(11) lastIndexOf()

public int lastIndexOf(char ch);

String s = "babbab";

sopln(s.LastIndexOf('a')); 4th index

If we want 3rd, 4th or any customized occurence we have to write code explicitly.

(12) toLowerCase()

public String toLowerCase();

(12) toUpperCase()

public String toUpperCase();

(13) trim() method & its use case:-

import java.util.*;

class Test

{

P S V main(String[] args)

{

Scanner SC = new Scanner(System.in);

sopln("Enter your city name");

```
String name = SC.nextLine().toLowerCase().
                                    trim();
if (name.equals("Hyderabadi"))
{
    Sopln("Hello Hyderabadi, Aadaab..");
}
else if (name.equals("Chennai"))
{
    Sopln("Hello Madrasi, Vanakkam...");
}
else if (name.equals("Bangalore"))
{
    Sopln("Hello Kannadiga, Namaskara...");
}
else
{
    Sopln("Please enter valid city name");
}
```

→ If user enter small or capital letter we can tackle it by using either —

i) toLowerCase() method.

ii) equalsIgnoreCase() method.

If user enter space either at beginning or end of input i.e. city name then we can tackle it by using —

trim() method.

trim() method can remove blank space only at beginning & end of input string not in

middle of string.

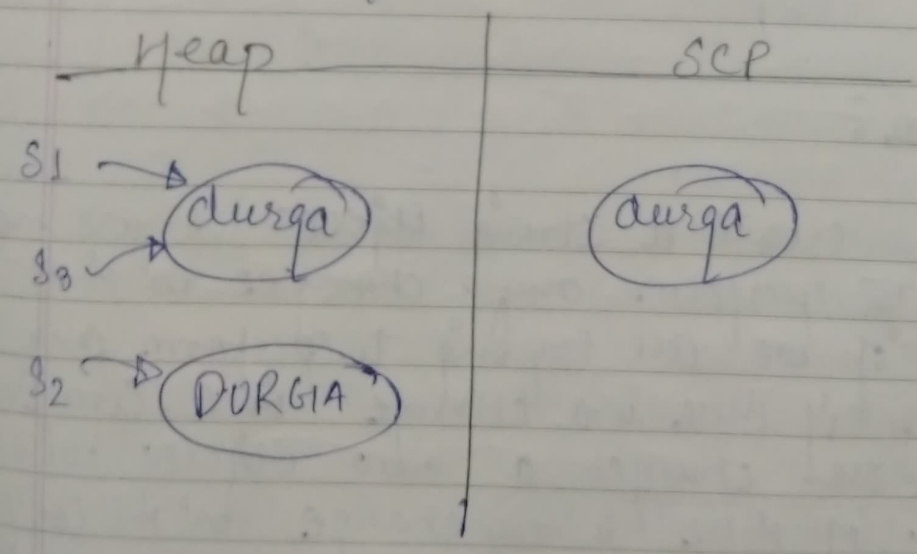
eg:- X-[Hydurabad]-X--

String s = "Durga Soft";
sopln(s.trim().length()); 10.
sopln(s.length()); 10 — same.

(7) Important conclusion about String immutability :-

Eg:- String s1 = new String("durga");
String s2 = s1.toUpperCase();
String s3 = s1.toLowerCase();

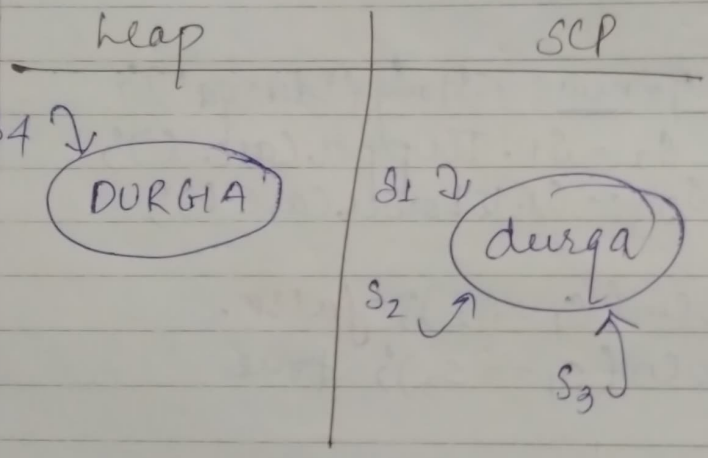
sopln(s1 == s2); false.
sopln(s1 == s3); true



eg2 :-

```
String s1 = "durga";
String s2 = s1.toString();
String s3 = s1.toLowerCase();
String s4 = s1.toUpperCase();

Sopln (s1 == s2); true
Sopln (s1 == s3); true
Sopln (s1 == s4); false
```



* Conclusion :-

Once we create a String Object we are not allow to perform any changes in that object, if we are trying to perform any changes, if there is a change in a content with those changes a new object will be created. if there is no change in the content existing object only will be reused, whether the existing object will be present in heap or SCP. area, rule is always same.

Ans (8) Creation of our own immutable class :-

```
final class Test
{
    private int i;

    Test() (int i)
    {
      this.i = i;
    }

    public Test modify(int i)
    {
      if( this.i == i)
        {
          return this;
        }
      else
        {
          return new Test(i);
        }
    }
}
```

String & all wrapper class declared as final because all are immutable in nature.

```
p s v main(String[] args)
{
  Test t1 = new Test(10);
  Test t2 = t1.modify(100);
  Test t3 = t1.modify(10);
```

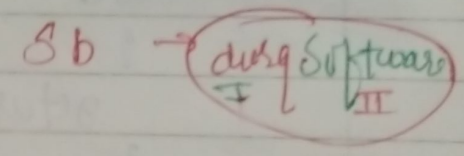
```
        sop(n (t1 == t2);
        sop(n (t1 == t3);
    }
}
```

(9) final Vs immutability :-

```
class Test
{
    P s v main(String[] args)
    {
        final StringBuffer sb = new StringBuffer("durga");
        sb.append("software");
        sop(n(sb);
        sb = new StringBuffer("ravi");
    }
}
```

durgasoftware

CE: Cannot assign a value to final variable sb.



→ We can't reassign to this reference variable to new object.

Note:- final terminology talks about variable but not for object.
immutability talks about object means its content but not about variable.

① Which of the following are meaningful?
final variable ✓

2) final object ✗
3) Immutable variable ✗
4) Immutable object ✓

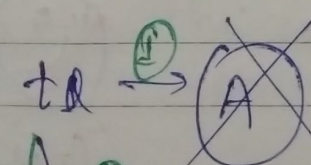
So, we cannot make StringBuffer method as immutable.

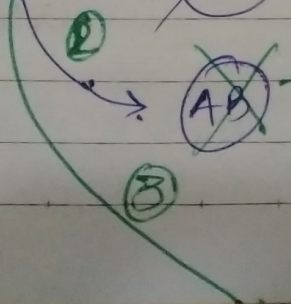
Practice Question & Explanation-1.

```
public class Test
{
    P s v main(String[] args)
    {
1)  String ta = "A";
2)  ta = ta.concat("B");
3)  String tb = "C";
4)  ta = ta.concat(tb);
5)  ta.replace('C', 'D');
6)  ta = ta.concat(tb);
7)  sopln(ta);
    }
}
```

What is result?

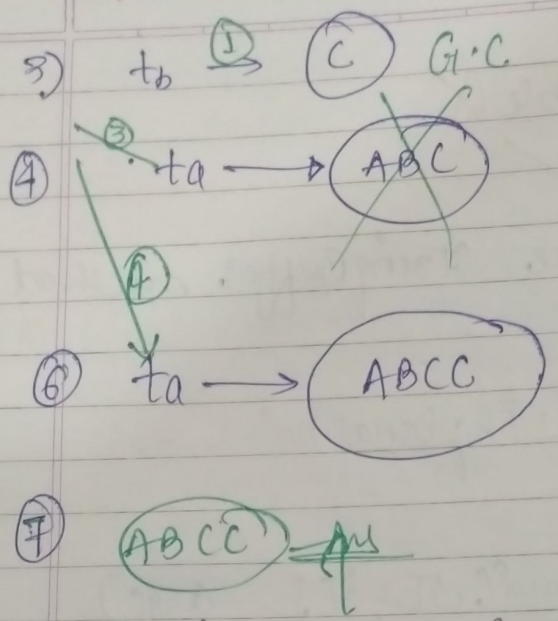
A) ABCD
B) ACD
C) ABCC ✓
D) ABD
E) ABDC

1)→ 

2) 

Explan: String objects are immutable. Hence, once we create a String object, we cannot perform any changes in that object.

491



* Practice question & Explanation - 2 :-

```
public class Test
{
    p s v main(String[] args)
    {
        1) String str = "  ";
        2) str.trim();
        3) Sopln(str.equals("") + "  " + str.isEmpty());
    }
}
```

(str.equals("") → false; str.isEmpty() → false)

What is the result?

A) true false
B) true true
C) false true
D) false false ✓

1) str → " " → 2 space characters

2) " " → No space due to trim

G.C

P. Ques -3

```
Public class Test
{
    P S V main(String[] args)
    {
        String s = "DURGA_SOFT";
        int len = s.trim().length();
        s.open(len);
    }
}
```

What is result?

s → DURGA_SOFT

A) 10 ✓
B) 9
C) 8
D) Compilation fails.

P. Ques -4

```
Public class Test
{
    P S V main(String[] args)
    {
        String s = "Hello world";
                    0 1 2 3 4 5
        s.trim();
        int i1 = s.indexOf("_");
        s.open(i1);
    }
}
```

What is result?

1) An exception is thrown at runtime

2) -1
3) 5
4) 0

Hello_World
0 1 2 3 4 5
5 →

Explanation:-
String objects are immutable. Once we create String object we cannot perform any changes in existing object.
If we are trying to perform any changes with those changes a new object will created.
trim() method will remove spaces present at beginning and end of string. But not middle blank spaces.
In the above example, trim() method won't remove the blank space present at 5th index, because it is present at middle of string.

P-5 Consider the following code:-

```
public class Test
{
    P s v main(String[] args)
    {
        String s1 = "Java";
        String s2 = new String("Java");
        // Line 1
        System.out.println("equal");
    }
```


3

A)
B)
C)
D)

(A)
Explan

s_2

s_2

== s1 → ○ ← s2

equals() considers case also

equalsIgnoreCase() ignores case method

```
    Sopln("Not equal");
  }
}
```

To print "equal" which code fragment should be inserted at Line-1.

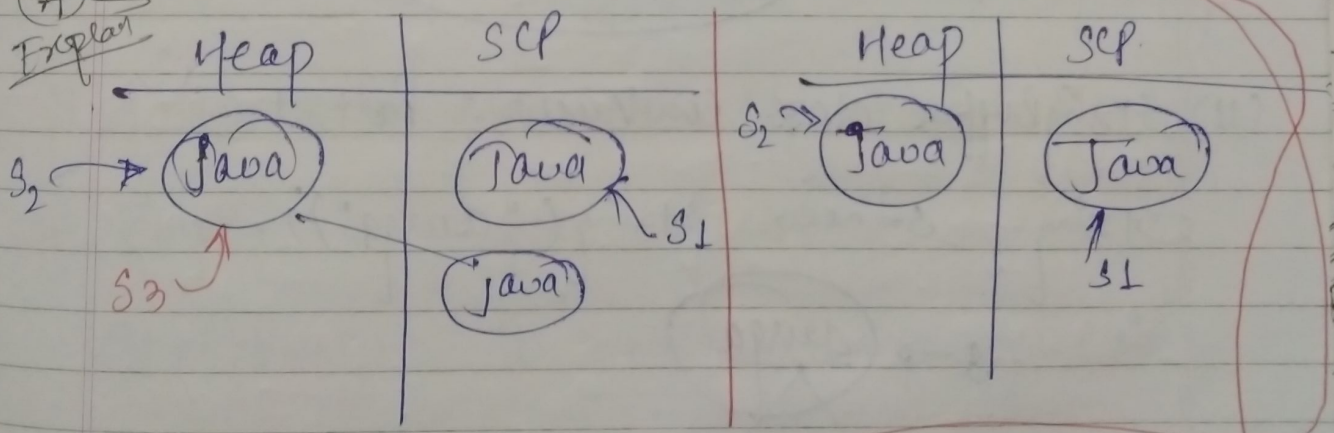
A) String s3 = s2;
if(s1 == s3) false.

B) if(s1.equalsIgnoreCase(s2)) true.

C) String s3 = s2;
if(s1.equals(s3)) — worry abt case also.

D) if(s1.toLowerCase() == s2.toLowerCase())
check address.

(A) → Explain



StringBuffer

(10) Need of StringBuffer :- All required changes will be performed in existing Objects only. for any small changes no new Objects will be created.

String :-

s → durga → durgasoft → durgasoftware → durgasoftware solutions

Memory waste
performance down

StringBuffer :-

Sb → durgasoftware solutions.

(11) StringBuffer class Constructors :- Part-1

String s = new String("durga");

s → durga

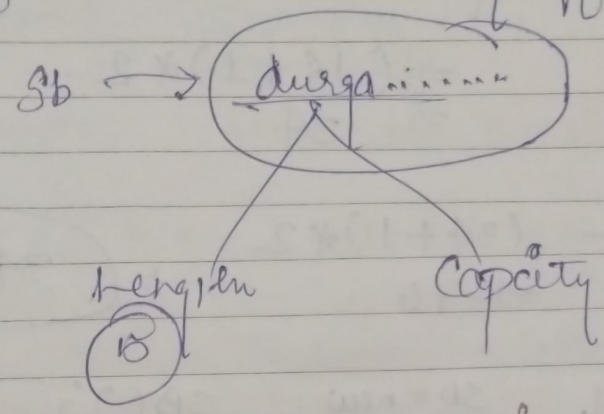
Length 5

Capacity 5

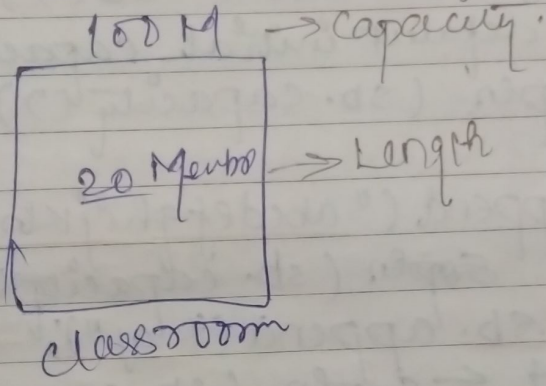
(11)

(1)

StringBuffer Sb = new StringBuffer("durga");



Eg:



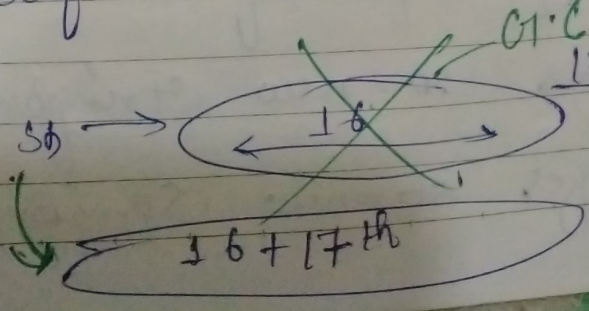
In String we never use word like Capacity. But in StringBuffer we can use Capacity.

(11) Important Constructors Of StringBuffer:

(1) Empty StringBuffer Objects:

StringBuffer Sb = new StringBuffer();
→ 16

An empty StringBuffer object can contain by default 16 character.



New capacity = (CC+1)*2 | formula.

⇒ (16+1)*2.
⇒ 34

→ (34+1)*2 → 34 + nth
⇒ 70

StringBuffer sb = new SB();
default initial capacity ⇒ 16.
sopln (sb.capacity()); 16

sb.append("abcdefghijklmnop");
sopln (sb.capacity()); 16 ✓
sb.append("q"); →1
(16+1)*2=34 ← sopln (sb.capacity()); 34
sb.append("abcdefghijklmnop");
sopln (sb.capacity());
sb.append("r");
sopln (sb.capacity()); 34
(34+1)*2 sb.append("s");
⇒ 70 sopln (sb.capacity()); 70

* Part-2 :-

If we would like to add 1000 characters, we can create a bigger String object at the begging only.

StringBuffer sb = new StringBuffer(int initialcapacity);

Eg:- StringBuffer sb = new StringBuffer(1000);

***

→ Part-3 for the given string I want to create equivalent StringBuffer.

StringBuffer sb = new StringBuffer(String s) ← given

Eg: SB sb = new SB("durga");
Sopln(sb.capacity()); 21

formula [Capacity = s.length() + 16]

(12) Important method of StringBuffer class:—

→ length(), capacity(), charAt(), setCharAt(), append().

(1) public int length(); → No. of characters present / How many characters in StrgBuffr
(2) public int capacity(); → Total characters can contain

(3) public char charAt(int index)
Eg: SB sb = new SB("durga"); [0-4]
Sopln(sb.charAt(3)); [g]
Sopln(sb.charAt(30));
RE: StringIndexOutOfBoundsException

(4) public void setCharAt(int index, char newchar);

SB sb = new SB("Java");
sb.setCharAt(0, 'Y'); Yava
Sopln(sb); Yava.

mahak
Date

In case of append data will be add at last position.

(5) public SB append(String s)
append(byte b)
(int i)
(long l)
(float f)
;

Overloaded methods

Ide can take any type of argument for append method.
Same method name but different parameter.

eg:- SB sb = new SB();
sb.append("PI value is");
String
sb.append(3.14);
double.
sb.append("It is exactly");
String.
sb.append(true);
boolean.
sop(sb);
PI value is 3.14It is exactly true.

⇒ insert(), delete(), deleteCharAt(), reverse().

(6) public SB insert(int index, String s);
(int index, double d)
" boolean b)
" char ch)
;
;
;
Overloaded method.

It is going to add content at our specified position.

eg: SB sb=new SB("abcdefgh");
sb.insert(2,"xyz");
sopln(sb); ab xyz cdefgh
0 1 2 3 4 ...

sb.insert(2, true); 2nd Index
↳ abtrue cdefgh.

sb.insert(2, 10.5);
ab10.5cdefgh.

⑦ delete:

public SB .delete(int begin, int end)

→ It deletes the characters from begin & end to end-1 index.

eg: StringBuffer sb=new SB("abcdefgh");
sb.delete(2, 5); → (2 to 4)
sopln(sb); abfgh.

ab cde fgh
delete

⑧ deleteCharAt()

public SB deleteCharAt(int index)

eg: SB sb=new SB("abcdefgh");
sb.deleteCharAt(3);
sopln(sb); abcefgh. ✓

Note:- In String No reverse method.

(9) reverse()

public SB reverse(). → reverse order of characters

eg:-

```
SB sb = new SB("durga");
sb.reverse();
sopln(sb); agrud. ✓
```

reverse the content of StringBuffer.

→ setLength(), ensureCapacity(), trimToSize().

(10) setLength()

public void setLength(int length)

```
SB sb = new SB("Aishwarya Abhi");
sb.setLength(8);
sopln(sb); Aishwarya.
```

We can set Length (decrease or increase) the StringBuffer content.

(11) ensureCapacity()

public void ensureCapacity(int capacity)

```
eg: SB sb = new SB();
sopln(sb.capacity()); 16
sb.ensureCapacity(1000);
sopln(sb.capacity()); 1000.
```

Based on our requirement if we want to increase capacity dynamically of SB. we use this method.

(12) trimToSize()

→ public void trimToSize()

```
SB sb = new SB(1000);
sb.append("ABC");
sopen(sb.capacity()); 1000
sb.trimToSize();
sopen(sb.capacity()); 3
```

remaining 997 memory allocation is deallocated.

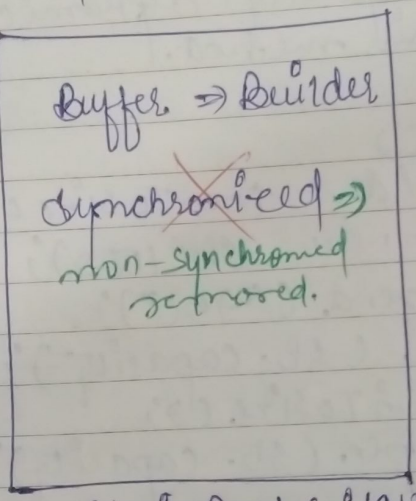
* StringBuilder — 1.5V

(1) Need of StringBuilder? We use this method to overcome from synchronized nature of that method (StringBuffer).

StringBuilder used in Multithreaded Environment. But StringBuffer not. due to its synchronized nature all methods of StringBuffer is synchronized in nature. means one thread can access its object at a time.

StringBuilder introduced in 1.5V.

(15) Difference b/w StringBuffer & StringBuilder.



StringBuffer.java
Builder.java

In StringBuffer all synchronized keyword is removed and becomes StringBuilder.

and, all & every method present in StringBuffer & every constructor also are exactly same as present in StringBuilder.

Only One difference is that it is non-synchronized in nature. nothing more than that.

At a time only one thread is allow to operate data consistency problem we should go for StringBuffer concept. &.

If Multiple thread is allowed to operate go for StringBuilder concept.

StringBuffer	StringBuilder
1) Most of the methods present inside a StringBuffer are synchronized	1) No method present inside StringBuilding is synchronized.
2) At a time only one thread is allowed operate on StringBuffer object & hence it is thread safe.	2) At a time multiple threads are allowed to operate on StringBuider object & hence it is not thread safe.
3) Thread are required to wait to operate on StringBuffer object & hence Relatively performance is slow	3) Threads are not required to wait to operate on StringBuilder object & hence relatively performace is high.
4) Introduced in 1.0 version	4) Introduced in 1.5 version

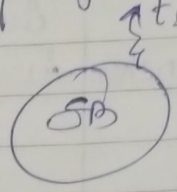
(15) String vs StringBuffer vs StringBuilder :-
Rather we should go for String, StringBuffer, StringBuilder concept.

1- If the content is always fixed wont change frequently go for String concept.

String

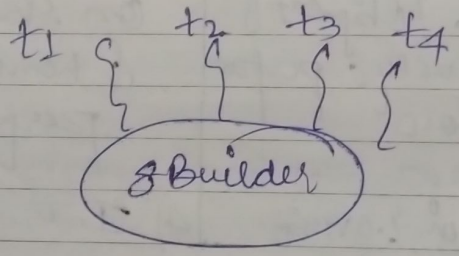
2- If the content is not fixed keep on changing but thread safety is required.

means only one thread is allowed at a time. go for StringBuffer.

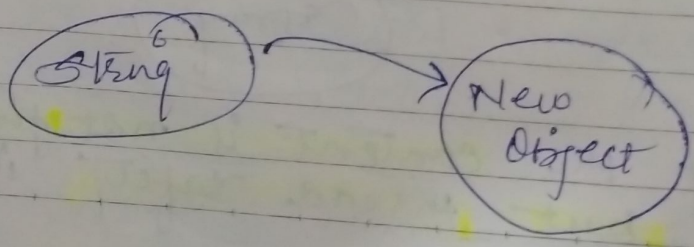


By StringBuffer Object is always thread safe.

(3) If content is not fixed. keep on changing & dont want thread safety. and means multiple thread allow to operate on object. go for StringBuilder.



Note: String object is thread safe or not.?
String is always thread safe - reason is that once we create a object of String we are not allow to change the content of that object. due to this change a new object will be created.



Not only String all immutable objects in Java, all wrapper class object, your own Immutable test class object. are by default thread safe bec= no one is allow to perform modification in the existing object.

(15) ~~Method~~

	String	StringBuffer	StringBuilder.
Object content	can't change	changeable	changeable.
thread safety	thread safe	~~not~~ thread safe	not thread safe.

Note: All immutable object in Java and all wrapper class are thread safe.

(16) Method = chaining :-

All the method present in String, StringBuffer, StringBuilder having same return types most of the time.

Eg: SB append()
SB insert()
SB delete()
SB reverse()

return type.

sb.m1().m2().m3()

↓

append, delete

- After applying a method on a result you can call another method. this is kyas method chaining.
- It is very common concept in no string sBuffer, sBuilder.

eg: All method will execute from left in method chaining.

```
SB sb = new SB();
sb.append("durga").append("solutions").reverse().
insert(2,"xyz").delete(3,7).....
sopln(sb);
```

snxtulosaqrud.

* Practice Questions - String & StringBuilder.

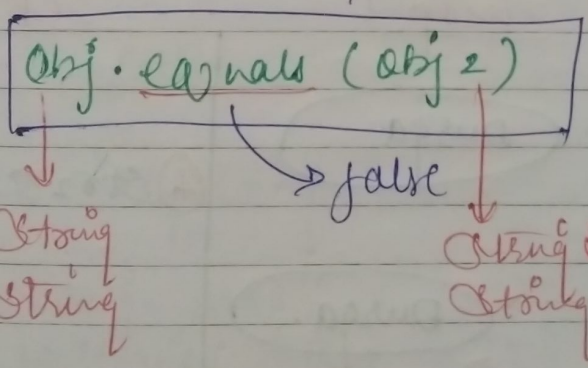
* Given the code fragment:-

```
public class Test
{
    p s v main(String[] args)
    {
        StringBuilder sb = new SB(5);
```

sb ⟶ (a, 5)

```
String s = "";  ---> s -> ("")
if (sb.equals(s)) -> false
{
    sopln("match 1");
}
else if (sb.toString().equals(s.toString())) true.
{
    sopln("Match 2");
}
else
{
    sopln("No match");
}
}
```

A) Match 1
B) Match 2 ✓
C) No match
D) NullPointerException is thrown at runtime.



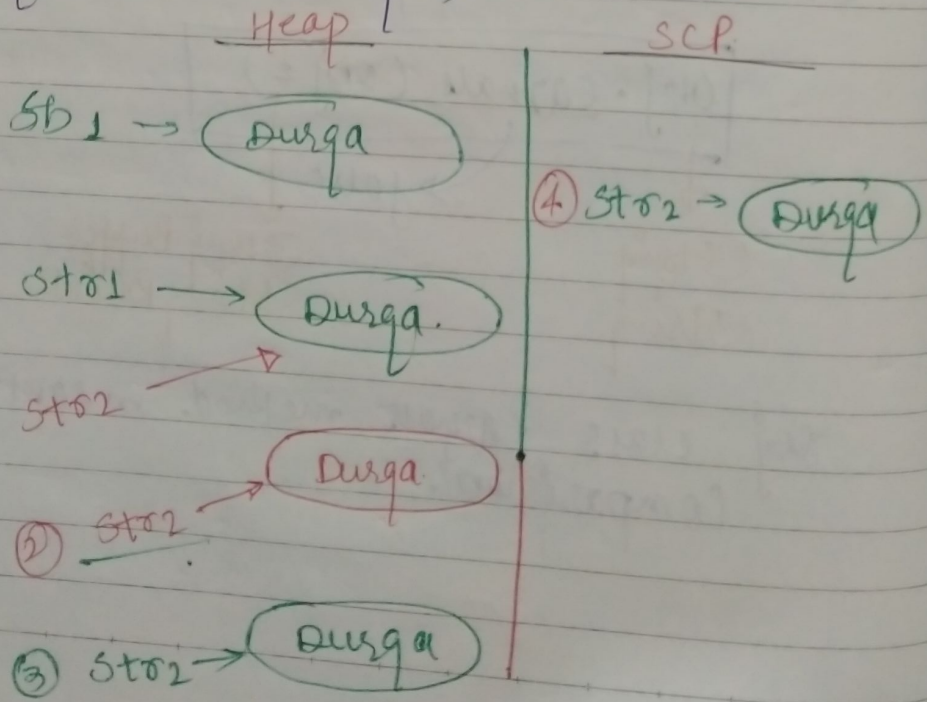
Obj class equals method meant for reference comparision.

2- Given code!

```
public class Test
{
    p s v main(String[] args)
    {
        StringBuilder sb1 = new SB("Durga");
        String str1 = sb1.toString();
        // Insert code => Line-1
        System.out.println(str1 == str2);
    }
}
```

Which code fragment, when inserted at Line-1, enables the code to print true?

A) `String str2 = str1;` ✓ true
B) `String str2 = new String(str1);` false
C) `String str2 = sb1.toString();`
D) `String str2 = "Durga";`



3- Given code:

Which statement will empty the contents of a StringBuilder varaible named sb?

A) sb.deleteAll() → method not exist for SB
b) sb.delete(0, sb.size()) – no size method for SB
c) sb.delete(0, sb.length()) ✓
d) sb.removeAll() – not exist.

sb.delete(begin, end)
removes content from begin index to n-1 index.

Eg:- SB sb = new SB("Durga");
sb.delete(0, 5);
↓
from 0 to n-1 ⇒ 0-(5)-4
⇒ 0-4 all character removed

D u r g a
0 1 2 3 4

4- Given code:

```
class MyString
{
    String msg;
    MyString(String msg)
    {
        this.msg = msg;
    }
}
```

no toString() method is written here.

```
public class Test
{
    p s v main(String[] args)
    {
        sopln("Hello" + new StringBuilder("Java SE 8"));
        sopln("Hello" + new MyString("Java SE 8"));
    }
}
```

What is the result?

A) Hello Java SE 8
Hello MyString@<hashcode>

B) Hello Java SE 8
Hello Java SE 8

C) Hello Java.lang.StringBuilder@<hashcode>
Hello myString@<hashcode>

D) Compilation Fails

Explanation:

```
Public class Test
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        sopln(t1);
    }
}
```

In this case Object class ToString() method will be called & give result as:

Classname@<Hashcode>.

eg:- Test@<5642001>

To overcome from this - we have to write our own toString() method.

```
Public String toString()
{
    return "Test Object";
}
```

O/p:- Test Object.

Our classes also

→ In I Highly recommended to write own toString() method.

In String, StringBuffer, all wrapper class toString() method already overridden for meaningful representation.

Q:- you are developing a banking module. you have developed a class name MaskTest that has a mask method.

Given Code:-

```
class MaskTest
{
```

```
public static String mask(String creditCard)
{
    String x = "XXXX-XXXX-XXXX-";
    // Line I
}
public static void main(String[] args)
{
    · Sopln (mask("1234-5678-9101-5979"));
}
```

Hide (1234, 5678, 9101) — display (5979)

→ You must ensure that mask method return a String that hides all digits of the credid card no. except last four digits and hyphens that seprate each group of 4-digits.

→ which two code fragment should you use at line-1 independetly to achieve the reqirement?

A) StringBuilder sb = new StringBuilder(creditcard); ↓ 1234-5979
sb.substring(15,19);
return x + sb;

not assign any string after variable

[XXXX-XXXX-XXXX- + 1234-5678-9101-5979] not worked

(B) return x + creditcard.substring(15,19);
↓ + 5979

Ans

C) StringBu...

D) StringB...

Explan...

① SCB.

5979

② rea...

x →

+

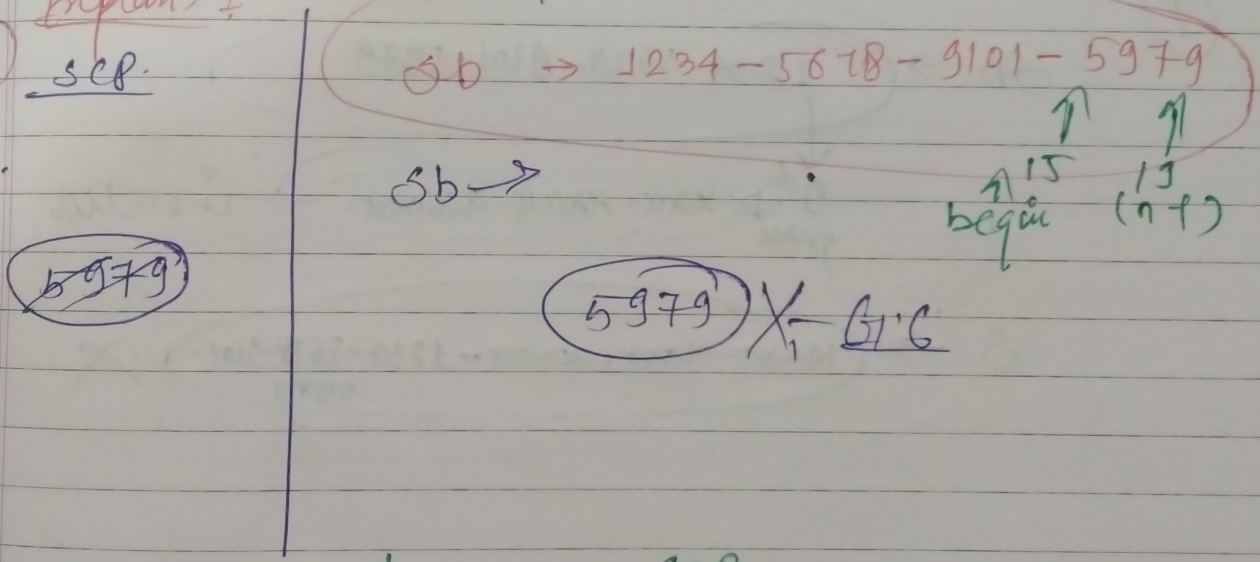
Credit card

c) StringBuilder sb = new StringBuilder(x);
sb.append(creditcard, 15, 19)
return sb.toString();
begin
end (n+1)

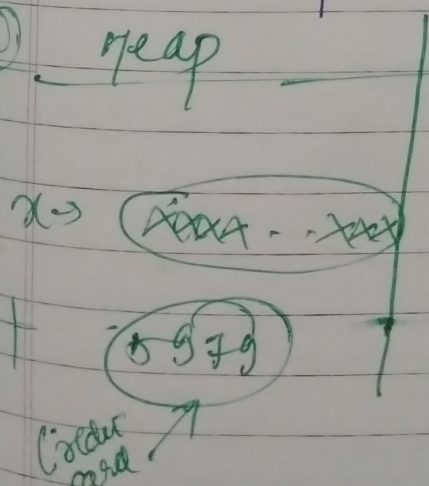
d) ~~StringBuilder~~ sb = new StringBuilder(creditcard);
StringBuilder s = sb.insert(0, x);
return s.toString();

Explain:

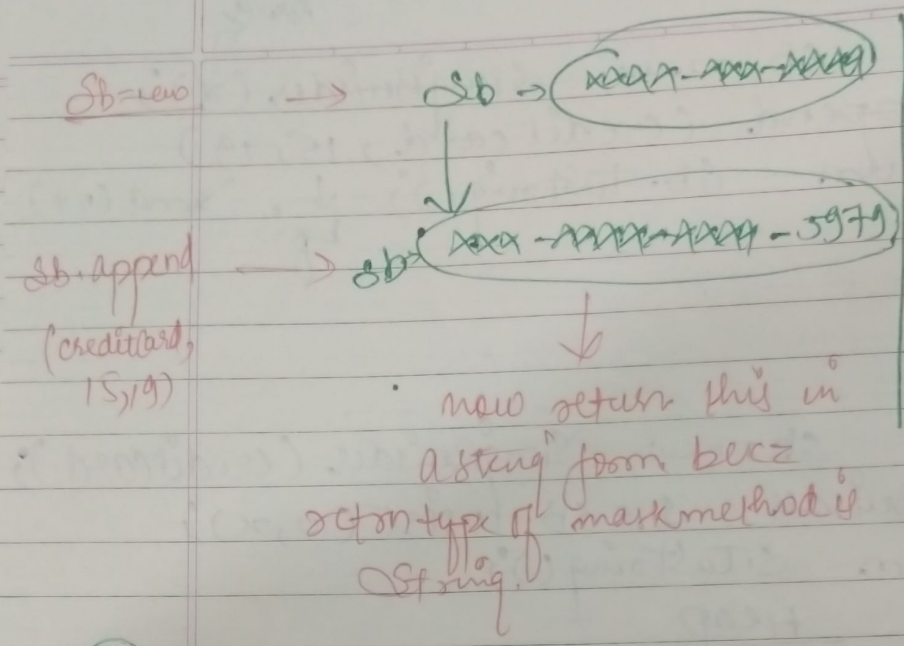
① SCP



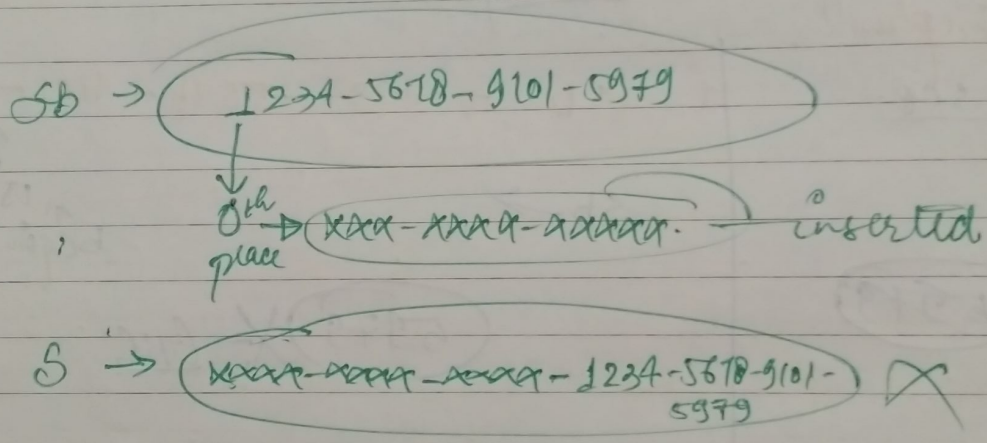
② Heap



SCP


Sb=new
Sb
xxxx-xxxx-xxxx
Sb.append
(creditcard,
15,19)
sb
xxxx-xxxxxxxxxx-5979
now return this in
a string form bcz
return type of mask method is
String.

(4)


Sb
1234-5678-9101-5979
8th
place
xxxx-xxxx-xxxxxx
inserted
S
xxxx-xxxx-xxxx-1234-5678-9101-
5979